In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```python
df = pd.read_csv("diamonds_new.csv")
df.head(19)
```

Out[ ]:

	carat	cut	color	clarity	table	x	y	z	price_new
0	0.23	Ideal	E	SI2	55.0	3.95	3.98	2.43	163.0
1	0.21	Premium	E	SI1	61.0	3.89	3.84	2.31	163.0
2	0.23	Good	E	VS1	65.0	4.05	4.07	2.31	163.5
3	0.29	Premium	I	VS2	58.0	4.20	4.23	2.63	167.0
4	0.31	Good	J	SI2	58.0	4.34	4.35	2.75	167.5
5	0.24	Very Good	J	VVS2	57.0	3.94	3.96	2.48	168.0
6	0.24	Very Good	I	VVS1	57.0	3.95	3.98	2.47	168.0
7	0.26	Very Good	H	SI1	55.0	4.07	4.11	2.53	168.5
8	0.22	Fair	E	VS2	61.0	3.87	3.78	2.49	168.5
9	0.23	Very Good	H	VS1	61.0	4.00	4.05	2.39	169.0
10	0.30	Good	J	SI1	55.0	4.25	4.28	2.73	169.5
11	0.23	Ideal	J	VS1	56.0	3.93	3.90	2.46	170.0
12	0.22	Premium	F	SI1	61.0	3.88	3.84	2.33	171.0
13	0.31	Ideal	J	SI2	54.0	4.35	4.37	2.71	172.0
14	0.20	Premium	E	SI2	62.0	3.79	3.75	2.27	172.5
15	0.32	Premium	E	I1	58.0	4.38	4.42	2.68	172.5
16	0.30	Ideal	I	SI2	54.0	4.31	4.34	2.68	174.0
17	0.30	Good	J	SI1	54.0	4.23	4.29	2.70	175.5
18	0.30	Good	J	SI1	56.0	4.23	4.26	2.71	175.5

In [ ]:
```python
df.info()
```

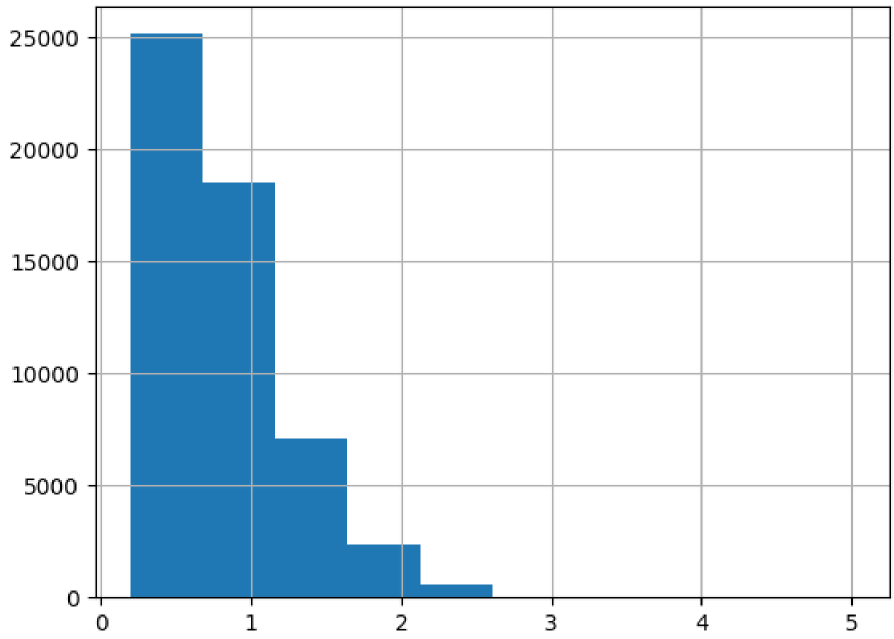
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   carat      53841 non-null  float64
 1   cut        53940 non-null  object 
 2   color      53884 non-null  object 
 3   clarity    53940 non-null  object 
 4   table      53877 non-null  float64
 5   x          53940 non-null  float64
 6   y          53940 non-null  float64
 7   z          53940 non-null  float64
 8   price_new  53940 non-null  float64
dtypes: float64(6), object(3)
memory usage: 3.7+ MB
```

In [ ]:
```python
df.isna().sum()
```

Out[ ]:
```
carat        99
cut           0
color        56
clarity       0
table        63
x             0
y             0
z             0
price_new     0
dtype: int64
```

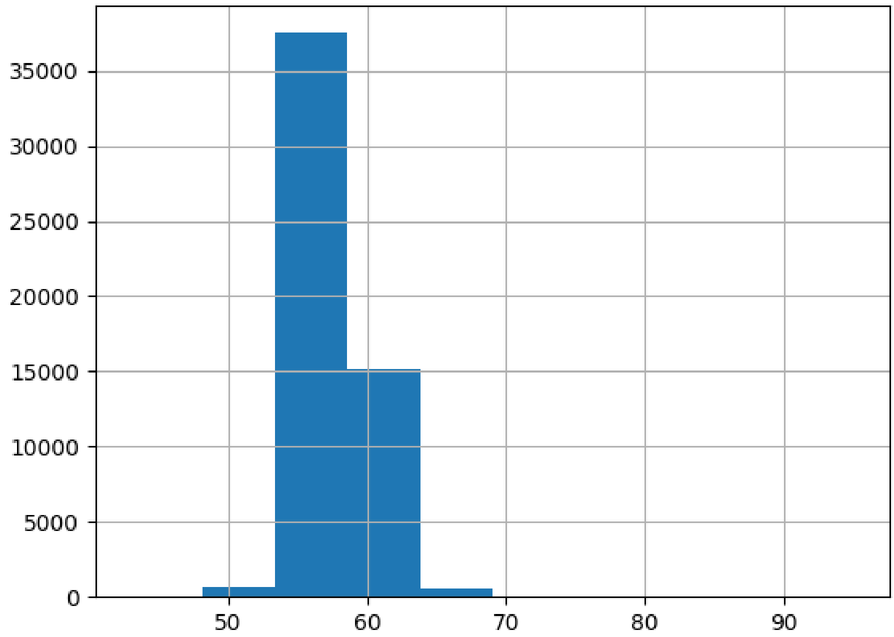
In [ ]:
```python
df['carat'].hist()
```

Out[ ]: <Axes: >



In [ ]:
```
df['table'].hist()
```

Out[ ]: <Axes: >



In [ ]:
```
df.describe()
```

Out[ ]:

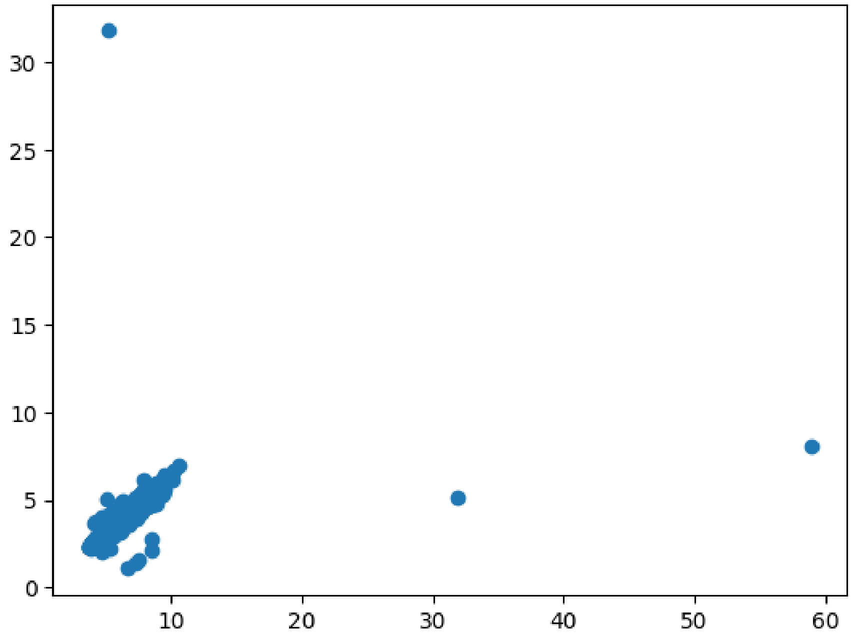
	carat	table	x	y	z	price_new
count	53841.000000	53877.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.798120	57.457719	5.731157	5.734526	3.539635	1966.399861
std	0.474428	2.235742	1.121761	1.142135	0.703869	1994.719869
min	0.200000	43.000000	0.000000	0.000000	0.000000	163.000000
25%	0.400000	56.000000	4.710000	4.720000	2.910000	475.000000
50%	0.700000	57.000000	5.700000	5.710000	3.530000	1200.500000
75%	1.040000	59.000000	6.540000	6.540000	4.040000	2662.125000
max	5.010000	95.000000	10.740000	58.900000	31.800000	9411.500000

In [ ]:
```
df[(df['x']==0) | (df['y']==0) | (df['z']==0)].index
```

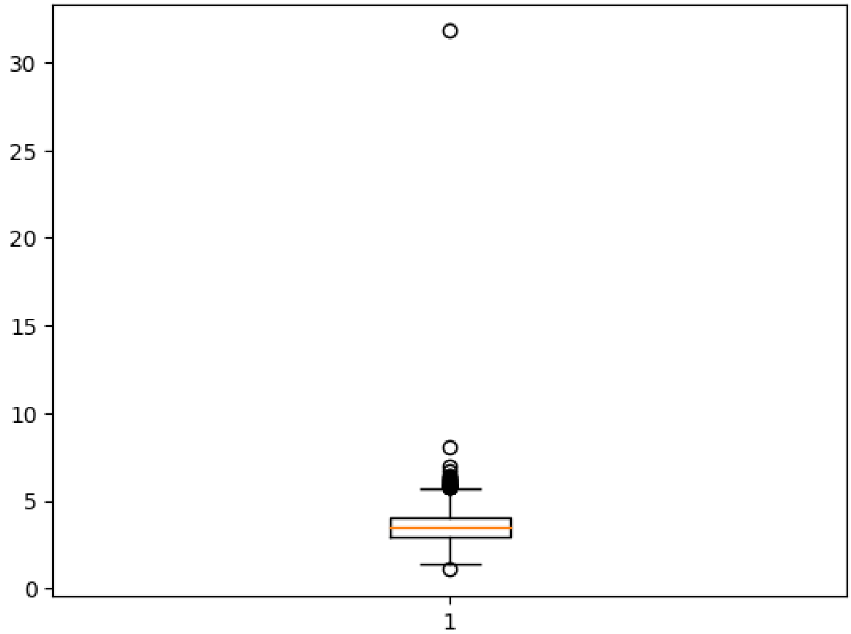
Out[ ]:
```
Index([11182, 11963, 15951, 24520, 26243, 27429, 49556, 49557], dtype='int64')
```

In [ ]:
```
df2 = df.drop([11182, 11963, 15951, 24520, 26243, 27429, 49556, 49557])
```

In [ ]:
```
plt.scatter(df2['y'], df2['z'])
plt.show()
```



In [ ]:
```
plt.boxplot(df2['z'])
plt.show()
```



In [ ]:
```
df2[(df2['y'] > 15) | (df2['z'] > 15)].index
```

Out[ ]:
```
Index([24067, 48410, 49189], dtype='int64')
```

In [ ]:
```
df3 = df2.drop([24067, 48410, 49189])
```

In [ ]:
```
df3.isna().sum()
```

Out[ ]:
```
carat        99
cut           0
color        56
clarity       0
table        63
x             0
y             0
z             0
price_new     0
dtype: int64
```

In [ ]:
```
df3['carat'].fillna(df3['carat'].median(), inplace=True)
df3['table'].fillna(df3['table'].median(), inplace=True)
df3['color'].fillna("G", inplace=True)
```

In [ ]:
```
df3.isna().sum()
```

```
Out[ ]: carat        0
        cut          0
        color        0
        clarity      0
        table        0
        x            0
        y            0
        z            0
        price_new    0
        dtype: int64
```

```
In [ ]: df3.columns
```

```
Out[ ]: Index(['carat', 'cut', 'color', 'clarity', 'table', 'x', 'y', 'z',
               'price_new'],
              dtype='object')
```

```
In [ ]: y = df3['price_new']
        x = df3.drop("price_new", axis=1)
```

```
In [ ]: x = pd.get_dummies(x)
```

```
In [ ]: from sklearn.preprocessing import MinMaxScaler
        from sklearn.model_selection import train_test_split
```

```
In [ ]: mn = MinMaxScaler()
        x = mn.fit_transform(x)
```

```
In [ ]: x_train, x_test, y_train , y_test = train_test_split(x, y, test_size=0.2, random_state= 134)
```

```
In [ ]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, Dropout
```

```
In [ ]: x.shape
```

```
Out[ ]: (53929, 25)
```

```
In [ ]: ### input = 25, hidden1 = 32, hidden2 = 16, hidden3 = 8, output = 1
```

```
In [ ]: model1 = Sequential()
        model1.add(Dense(32, activation = "relu", input_shape = (25,)))
        model1.add(Dense(16, activation = "relu"))
        model1.add(Dense(8, activation = "relu"))
        model1.add(Dense(1, activation = None))
        model1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_24 (Dense)            (None, 32)                832

 dense_25 (Dense)            (None, 16)                528

 dense_26 (Dense)            (None, 8)                 136

 dense_27 (Dense)            (None, 1)                 9

=================================================================
Total params: 1,505
Trainable params: 1,505
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: model1.compile(optimizer = "sgd", loss = "mean_squared_error", metrics = ["mean_squared_error"])
```

```
In [ ]: model1.fit(x_train, y_train, epochs = 10, batch_size = 128)
```

```
Epoch 1/10
338/338 [==============================] - 1s 2ms/step - loss: 1406556309285009138647040.0000 - mean_squared_err
or: 1406556309285009138647040.0000
Epoch 2/10
338/338 [==============================] - 0s 1ms/step - loss: 18260890977566720.0000 - mean_squared_error: 1826
0890977566720.0000
Epoch 3/10
338/338 [==============================] - 0s 1ms/step - loss: 21400383488.0000 - mean_squared_error: 2140038348
8.0000
Epoch 4/10
338/338 [==============================] - 0s 1ms/step - loss: 4023167.2500 - mean_squared_error: 4023167.2500
Epoch 5/10
338/338 [==============================] - 0s 1ms/step - loss: 3998242.5000 - mean_squared_error: 3998242.5000
Epoch 6/10
338/338 [==============================] - 0s 1ms/step - loss: 3998219.7500 - mean_squared_error: 3998219.7500
Epoch 7/10
338/338 [==============================] - 0s 1ms/step - loss: 3997850.5000 - mean_squared_error: 3997850.5000
Epoch 8/10
338/338 [==============================] - 0s 1ms/step - loss: 3998243.5000 - mean_squared_error: 3998243.5000
Epoch 9/10
338/338 [==============================] - 0s 1ms/step - loss: 3998069.0000 - mean_squared_error: 3998069.0000
Epoch 10/10
338/338 [==============================] - 0s 1ms/step - loss: 3997983.7500 - mean_squared_error: 3997983.7500
```

Out[ ]:
```
<keras.callbacks.History at 0x7f53246d9b70>
```

In [ ]:
```
model1.evaluate(x_test, y_test)
```

```
338/338 [==============================] - 1s 1ms/step - loss: 3895393.2500 - mean_squared_error: 3895393.2500
```

Out[ ]:
```
[3895393.25, 3895393.25]
```

model2

In [ ]:
```
model2 = Sequential()
model2.add(Dense(64, input_shape = (25,), activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(64, activation = "relu"))
model2.add(Dense(1, activation = None))
```

In [ ]:
```
model2.compile(optimizer = "sgd", loss = "mean_squared_error", metrics = ["mean_squared_error"])
model2.fit(x_train, y_train, epochs = 30, batch_size = 128)
```

```
Epoch 1/30
338/338 [==============================] - 2s 3ms/step - loss: nan - mean_squared_error: nan
Epoch 2/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 3/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 4/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 5/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 6/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 7/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 8/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 9/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 10/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 11/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 12/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 13/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 14/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 15/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 16/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 17/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 18/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 19/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 20/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 21/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 22/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 23/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 24/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 25/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 26/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 27/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 28/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 29/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
Epoch 30/30
338/338 [==============================] - 1s 2ms/step - loss: nan - mean_squared_error: nan
```

Out[ ]: `<keras.callbacks.History at 0x7f53245571c0>`

Model 3

In [ ]:
```
model3 = Sequential()
model3.add(Dense(128, input_shape = (25,), activation = "relu"))
model3.add(Dense(128, activation = "relu"))
model3.add(Dropout(0.15))
model3.add(Dense(128, activation = "relu"))
model3.add(Dropout(0.15))
model3.add(Dense(64, activation = "relu"))
model3.add(Dropout(0.15))
model3.add(Dense(64, activation = "relu"))
model3.add(Dropout(0.15))
model3.add(Dense(32, activation = "relu"))
model3.add(Dense(16, activation = "relu"))
model3.add(Dense(1, activation = None))
```

In [ ]:
```
model3.compile(optimizer = "adam", loss = "mean_squared_error", metrics = ["mean_squared_error"])
model3.fit(x_train, y_train, epochs = 30, batch_size = 128)
```

```
Epoch 1/30
338/338 [==============================] - 3s 5ms/step - loss: 2536666.0000 - mean_squared_error: 2536666.0000
Epoch 2/30
338/338 [==============================] - 1s 3ms/step - loss: 227813.1719 - mean_squared_error: 227813.1719
Epoch 3/30
338/338 [==============================] - 1s 3ms/step - loss: 208016.6719 - mean_squared_error: 208016.6719
Epoch 4/30
338/338 [==============================] - 1s 3ms/step - loss: 196001.7188 - mean_squared_error: 196001.7188
Epoch 5/30
338/338 [==============================] - 1s 3ms/step - loss: 190584.9062 - mean_squared_error: 190584.9062
Epoch 6/30
338/338 [==============================] - 1s 3ms/step - loss: 189123.5312 - mean_squared_error: 189123.5312
Epoch 7/30
338/338 [==============================] - 1s 3ms/step - loss: 190394.3125 - mean_squared_error: 190394.3125
Epoch 8/30
338/338 [==============================] - 1s 4ms/step - loss: 180359.5625 - mean_squared_error: 180359.5625
Epoch 9/30
338/338 [==============================] - 1s 3ms/step - loss: 182941.4844 - mean_squared_error: 182941.4844
Epoch 10/30
338/338 [==============================] - 1s 3ms/step - loss: 183868.1250 - mean_squared_error: 183868.1250
Epoch 11/30
338/338 [==============================] - 1s 3ms/step - loss: 182024.2500 - mean_squared_error: 182024.2500
Epoch 12/30
338/338 [==============================] - 1s 4ms/step - loss: 175677.3594 - mean_squared_error: 175677.3594
Epoch 13/30
338/338 [==============================] - 1s 3ms/step - loss: 176947.3281 - mean_squared_error: 176947.3281
Epoch 14/30
338/338 [==============================] - 1s 3ms/step - loss: 175572.0469 - mean_squared_error: 175572.0469
Epoch 15/30
338/338 [==============================] - 1s 4ms/step - loss: 176571.2344 - mean_squared_error: 176571.2344
Epoch 16/30
338/338 [==============================] - 1s 3ms/step - loss: 171691.3281 - mean_squared_error: 171691.3281
Epoch 17/30
338/338 [==============================] - 1s 3ms/step - loss: 174111.7344 - mean_squared_error: 174111.7344
Epoch 18/30
338/338 [==============================] - 1s 3ms/step - loss: 172084.6406 - mean_squared_error: 172084.6406
Epoch 19/30
338/338 [==============================] - 1s 4ms/step - loss: 172675.3438 - mean_squared_error: 172675.3438
Epoch 20/30
338/338 [==============================] - 1s 3ms/step - loss: 172956.4375 - mean_squared_error: 172956.4375
Epoch 21/30
338/338 [==============================] - 1s 4ms/step - loss: 175000.2500 - mean_squared_error: 175000.2500
Epoch 22/30
338/338 [==============================] - 1s 3ms/step - loss: 167298.3750 - mean_squared_error: 167298.3750
Epoch 23/30
338/338 [==============================] - 1s 3ms/step - loss: 164625.5938 - mean_squared_error: 164625.5938
Epoch 24/30
338/338 [==============================] - 1s 4ms/step - loss: 165128.1250 - mean_squared_error: 165128.1250
Epoch 25/30
338/338 [==============================] - 1s 3ms/step - loss: 161874.3906 - mean_squared_error: 161874.3906
Epoch 26/30
338/338 [==============================] - 1s 4ms/step - loss: 156333.6094 - mean_squared_error: 156333.6094
Epoch 27/30
338/338 [==============================] - 1s 3ms/step - loss: 159233.0625 - mean_squared_error: 159233.0625
Epoch 28/30
338/338 [==============================] - 1s 3ms/step - loss: 161281.2344 - mean_squared_error: 161281.2344
Epoch 29/30
338/338 [==============================] - 1s 3ms/step - loss: 153852.4531 - mean_squared_error: 153852.4531
Epoch 30/30
338/338 [==============================] - 1s 3ms/step - loss: 148845.4062 - mean_squared_error: 148845.4062
```

Out[ ]:

```
<keras.callbacks.History at 0x7f5324329960>
```

In [ ]:

```
model3.evaluate(x_test, y_test)
```

```
338/338 [==============================] - 1s 1ms/step - loss: 180751.6875 - mean_squared_error: 180751.6875
```

Out[ ]:

```
[180751.6875, 180751.6875]
```

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [ ]:

```
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[ ]:

▾ LinearRegression

LinearRegression()

In [ ]:

```
y_pred = lr.predict(x_test)
mean_squared_error(y_test, y_pred)
```

Out[ ]: 310612.43175525905

Assignment Begins

In [ ]:
```
df4 = df3.copy()
```

Label Encoding features based on their quality

In [ ]:
```
cut_worst_to_best = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']
clarity_worst_to_best = ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF']
color_worst_to_best = ['J','I','H','G','F','E','D']
```

In [ ]:
```
def manual_label_encode(x, list_):
    if x in list_:
        return list_.index(x)
```

In [ ]:
```
df4['cut'] = df4['cut'].apply(lambda x: manual_label_encode(x, cut_worst_to_best))
df4['clarity'] = df4['clarity'].apply(lambda x: manual_label_encode(x, clarity_worst_to_best))
df4['color'] = df4['color'].apply(lambda x: manual_label_encode(x, color_worst_to_best))
```

In [ ]:
```
df4
```

Out[ ]:

	carat	cut	color	clarity	table	x	y	z	price_new
0	0.23	4	5	1	55.0	3.95	3.98	2.43	163.0
1	0.21	3	5	2	61.0	3.89	3.84	2.31	163.0
2	0.23	1	5	4	65.0	4.05	4.07	2.31	163.5
3	0.29	3	1	3	58.0	4.20	4.23	2.63	167.0
4	0.31	1	0	1	58.0	4.34	4.35	2.75	167.5
...	...	...	...	...	...	...	...	...	...
53935	0.72	4	6	2	57.0	5.75	5.76	3.50	1378.5
53936	0.72	1	6	2	55.0	5.69	5.75	3.61	1378.5
53937	0.70	2	6	2	60.0	5.66	5.68	3.56	1378.5
53938	0.86	3	2	1	58.0	6.15	6.12	3.74	1378.5
53939	0.75	4	6	1	55.0	5.83	5.87	3.64	1378.5

53929 rows × 9 columns

In [ ]:
```
df4.corr()
```

Out[ ]:

	carat	cut	color	clarity	table	x	y	z	price_new
carat	1.000000	-0.135033	-0.291413	-0.352836	0.181658	0.977764	0.976844	0.976031	0.921604
cut	-0.135033	1.000000	0.020516	0.189196	-0.433310	-0.126281	-0.125909	-0.152495	-0.053567
color	-0.291413	0.020516	1.000000	-0.025718	-0.026475	-0.270748	-0.270555	-0.274892	-0.172532
clarity	-0.352836	0.189196	-0.025718	1.000000	-0.160388	-0.372973	-0.367635	-0.376446	-0.146838
table	0.181658	-0.433310	-0.026475	-0.160388	1.000000	0.196129	0.189976	0.155850	0.127161
x	0.977764	-0.126281	-0.270748	-0.372973	0.196129	1.000000	0.998658	0.990758	0.887216
y	0.976844	-0.125909	-0.270555	-0.367635	0.189976	0.998658	1.000000	0.990420	0.888812
z	0.976031	-0.152495	-0.274892	-0.376446	0.155850	0.990758	0.990420	1.000000	0.881725
price_new	0.921604	-0.053567	-0.172532	-0.146838	0.127161	0.887216	0.888812	0.881725	1.000000

creating new training and testing sets

In [ ]:
```
y = df4['price_new']
x = df4.drop("price_new", axis=1)

minmax = MinMaxScaler(feature_range=(0,1))
x = minmax.fit_transform(x)
x_train, x_test, y_train , y_test = train_test_split(x, y, test_size=0.2, random_state= 123)
```

In [ ]:
```
from tensorflow.keras.regularizers import l2, l1, l1_l2
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import RMSprop
```

In [ ]:
```
model4 = Sequential()
model4.add(Dense(128, input_shape = (8,), activation = "relu", kernel_regularizer=l2(0.02)))
model4.add(Dense(128, activation = "relu", kernel_regularizer=l2(0.02)))
model4.add(Dropout(0.2))
model4.add(Dense(64, activation = "relu", kernel_regularizer=l2(0.02)))
model4.add(Dense(64, activation = "relu", kernel_regularizer=l2(0.02)))
model4.add(Dropout(0.15))
model4.add(Dense(32, activation = "relu", kernel_regularizer=l2(0.02)))
model4.add(Dropout(0.1))
model4.add(Dense(16, activation = "relu", kernel_regularizer=l2(0.01)))
model4.add(Dropout(0.1))
model4.add(Dense(8, activation = "relu", kernel_regularizer=l2(0.01)))
model4.add(Dense(1, activation = 'linear'))
model4.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_44 (Dense)            (None, 128)               1152      
                                                                 
 dense_45 (Dense)            (None, 128)               16512     
                                                                 
 dropout_8 (Dropout)         (None, 128)               0         
                                                                 
 dense_46 (Dense)            (None, 64)                8256      
                                                                 
 dense_47 (Dense)            (None, 64)                4160      
                                                                 
 dropout_9 (Dropout)         (None, 64)                0         
                                                                 
 dense_48 (Dense)            (None, 32)                2080      
                                                                 
 dropout_10 (Dropout)        (None, 32)                0         
                                                                 
 dense_49 (Dense)            (None, 16)                528       
                                                                 
 dropout_11 (Dropout)        (None, 16)                0         
                                                                 
 dense_50 (Dense)            (None, 8)                 136       
                                                                 
 dense_51 (Dense)            (None, 1)                 9         
                                                                 
=================================================================
Total params: 32,833
Trainable params: 32,833
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model4.compile(optimizer = "adam", loss = "mean_squared_error", metrics = ["mean_squared_error"])
model4.fit(x_train, y_train, epochs = 50, batch_size = 128, validation_data=(x_test, y_test), callbacks=[EarlyS
```

```
Epoch 1/50
338/338 [==============================] - 3s 5ms/step - loss: 2158404.0000 - mean_squared_error: 2158395.7500 - val_loss: 261434.3906 - val_mean_squared_error: 261424.9844
Epoch 2/50
338/338 [==============================] - 1s 3ms/step - loss: 364880.5625 - mean_squared_error: 364870.9688 - val_loss: 210681.0625 - val_mean_squared_error: 210670.9844
Epoch 3/50
338/338 [==============================] - 1s 3ms/step - loss: 315590.0312 - mean_squared_error: 315579.4688 - val_loss: 166737.7969 - val_mean_squared_error: 166726.8906
Epoch 4/50
338/338 [==============================] - 1s 3ms/step - loss: 289890.2812 - mean_squared_error: 289878.9062 - val_loss: 205623.7500 - val_mean_squared_error: 205611.8125
Epoch 5/50
338/338 [==============================] - 1s 3ms/step - loss: 263969.6875 - mean_squared_error: 263957.2500 - val_loss: 167066.2812 - val_mean_squared_error: 167053.5469
Epoch 6/50
338/338 [==============================] - 1s 3ms/step - loss: 246183.9531 - mean_squared_error: 246171.0156 - val_loss: 148185.9844 - val_mean_squared_error: 148172.8594
Epoch 7/50
338/338 [==============================] - 1s 3ms/step - loss: 244706.8281 - mean_squared_error: 244693.5156 - val_loss: 143354.9844 - val_mean_squared_error: 143341.3125
Epoch 8/50
338/338 [==============================] - 1s 3ms/step - loss: 235339.6562 - mean_squared_error: 235325.7031 - val_loss: 155573.5312 - val_mean_squared_error: 155559.3125
Epoch 9/50
338/338 [==============================] - 1s 3ms/step - loss: 224851.9844 - mean_squared_error: 224837.6094 - val_loss: 114327.8281 - val_mean_squared_error: 114313.1250
Epoch 10/50
338/338 [==============================] - 1s 3ms/step - loss: 222310.7500 - mean_squared_error: 222295.7969 - val_loss: 165172.4844 - val_mean_squared_error: 165157.3125
Epoch 11/50
338/338 [==============================] - 1s 3ms/step - loss: 216388.2500 - mean_squared_error: 216372.7656 - val_loss: 115445.3906 - val_mean_squared_error: 115429.6406
Epoch 12/50
338/338 [==============================] - 1s 3ms/step - loss: 203169.3281 - mean_squared_error: 203153.3594 - val_loss: 254586.3281 - val_mean_squared_error: 254570.0938
Epoch 13/50
338/338 [==============================] - 1s 3ms/step - loss: 204479.3438 - mean_squared_error: 204463.0156 - val_loss: 179704.5156 - val_mean_squared_error: 179687.8594
Epoch 14/50
338/338 [==============================] - 1s 3ms/step - loss: 199702.2812 - mean_squared_error: 199685.3438 - val_loss: 145884.5312 - val_mean_squared_error: 145867.3906
Epoch 15/50
338/338 [==============================] - 1s 3ms/step - loss: 191384.0312 - mean_squared_error: 191366.7188 - val_loss: 328804.4062 - val_mean_squared_error: 328787.0312
Epoch 16/50
338/338 [==============================] - 1s 3ms/step - loss: 194190.2031 - mean_squared_error: 194172.4375 - val_loss: 388515.0312 - val_mean_squared_error: 388496.9375
Epoch 17/50
338/338 [==============================] - 1s 3ms/step - loss: 195870.2969 - mean_squared_error: 195851.9062 - val_loss: 272791.6250 - val_mean_squared_error: 272773.0938
Epoch 18/50
338/338 [==============================] - 1s 3ms/step - loss: 186613.4062 - mean_squared_error: 186594.8125 - val_loss: 142228.2031 - val_mean_squared_error: 142209.1250
Epoch 19/50
338/338 [==============================] - 1s 3ms/step - loss: 185611.9219 - mean_squared_error: 185592.6094 - val_loss: 227830.9062 - val_mean_squared_error: 227811.3594
```

Out[ ]: <keras.callbacks.History at 0x7f536c207a30>

Model Evaluation

In [ ]:
```
pred = model4.predict(x_test)
model4.evaluate(x_test, y_test)
```

```
338/338 [==============================] - 1s 2ms/step
338/338 [==============================] - 1s 2ms/step - loss: 114327.8047 - mean_squared_error: 114313.1250
```

Out[ ]: [114327.8046875, 114313.125]

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [ ]:
```
print("Mean Squared error is :", round(mean_squared_error(y_test, pred),2))
print("Mean Absolute error is :", round(mean_absolute_error(y_test, pred),2))
```

```
Mean Squared error is : 114313.12
Mean Absolute error is : 188.22
```

In [ ]:
```
model5 = Sequential()
model5.add(Dense(128, input_shape = (8,), activation = "relu", kernel_regularizer=l1_l2(0.05)))
model5.add(Dense(128, activation = "relu", kernel_regularizer=l1_l2(0.04)))
model5.add(Dropout(0.1))
model5.add(Dense(64, activation = "relu", kernel_regularizer=l1_l2(0.04)))
```

```
model5.add(Dropout(0.1))
model5.add(Dense(32, activation = "relu", kernel_regularizer=l1_l2(0.04)))
model5.add(Dense(16, activation = "relu", kernel_regularizer=l1_l2(0.02)))
model5.add(Dropout(0.05))
model5.add(Dense(8, activation = "relu", kernel_regularizer=l1_l2(0.02)))
model5.add(Dense(1, activation = 'linear'))
model5.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_67 (Dense)            (None, 128)               1152

 dense_68 (Dense)            (None, 128)               16512

 dropout_18 (Dropout)        (None, 128)               0

 dense_69 (Dense)            (None, 64)                8256

 dropout_19 (Dropout)        (None, 64)                0

 dense_70 (Dense)            (None, 32)                2080

 dense_71 (Dense)            (None, 16)                528

 dropout_20 (Dropout)        (None, 16)                0

 dense_72 (Dense)            (None, 8)                 136

 dense_73 (Dense)            (None, 1)                 9

=================================================================
Total params: 28,673
Trainable params: 28,673
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model5.compile(optimizer = RMSprop(learning_rate=0.01), loss = "mean_squared_error", metrics = ["mean_squared_e
model5.fit(x_train, y_train, epochs = 50, batch_size = 128, validation_data=(x_test, y_test), callbacks=[EarlyS
```

```
Epoch 1/50
338/338 [==============================] - 2s 4ms/step - loss: 1067213.6250 - mean_squared_error: 1067118.1250 -
val_loss: 841478.4375 - val_mean_squared_error: 841373.6250
Epoch 2/50
338/338 [==============================] - 1s 3ms/step - loss: 539767.9375 - mean_squared_error: 539654.6875 - v
al_loss: 275639.8438 - val_mean_squared_error: 275521.7500
Epoch 3/50
338/338 [==============================] - 1s 3ms/step - loss: 457866.7812 - mean_squared_error: 457739.8438 - v
al_loss: 2262308.5000 - val_mean_squared_error: 2262171.0000
Epoch 4/50
338/338 [==============================] - 1s 3ms/step - loss: 389034.4375 - mean_squared_error: 388896.5000 - v
al_loss: 825687.6875 - val_mean_squared_error: 825549.6250
Epoch 5/50
338/338 [==============================] - 1s 3ms/step - loss: 356404.9062 - mean_squared_error: 356263.6875 - v
al_loss: 540858.5625 - val_mean_squared_error: 540715.7500
Epoch 6/50
338/338 [==============================] - 1s 3ms/step - loss: 317830.2500 - mean_squared_error: 317684.5312 - v
al_loss: 224519.3750 - val_mean_squared_error: 224370.5156
Epoch 7/50
338/338 [==============================] - 1s 3ms/step - loss: 300612.3750 - mean_squared_error: 300462.6875 - v
al_loss: 549237.7500 - val_mean_squared_error: 549088.7500
Epoch 8/50
338/338 [==============================] - 1s 3ms/step - loss: 291538.0938 - mean_squared_error: 291388.0000 - v
al_loss: 760498.8750 - val_mean_squared_error: 760350.3125
Epoch 9/50
338/338 [==============================] - 1s 3ms/step - loss: 272749.5312 - mean_squared_error: 272598.9062 - v
al_loss: 843131.1875 - val_mean_squared_error: 842976.0625
Epoch 10/50
338/338 [==============================] - 1s 3ms/step - loss: 265617.8438 - mean_squared_error: 265466.4688 - v
al_loss: 373518.6875 - val_mean_squared_error: 373368.6562
Epoch 11/50
338/338 [==============================] - 1s 3ms/step - loss: 258326.1719 - mean_squared_error: 258174.4531 - v
al_loss: 379713.8750 - val_mean_squared_error: 379562.3750
Epoch 12/50
338/338 [==============================] - 1s 3ms/step - loss: 253241.0938 - mean_squared_error: 253088.9531 - v
al_loss: 103684.5547 - val_mean_squared_error: 103532.1797
Epoch 13/50
338/338 [==============================] - 1s 3ms/step - loss: 239481.6094 - mean_squared_error: 239329.6094 - v
al_loss: 138966.7188 - val_mean_squared_error: 138814.6875
Epoch 14/50
338/338 [==============================] - 1s 3ms/step - loss: 235599.7969 - mean_squared_error: 235448.7031 - v
al_loss: 326605.0312 - val_mean_squared_error: 326456.4062
Epoch 15/50
338/338 [==============================] - 1s 3ms/step - loss: 237873.8438 - mean_squared_error: 237725.2500 - v
al_loss: 421889.5625 - val_mean_squared_error: 421744.2500
Epoch 16/50
338/338 [==============================] - 1s 3ms/step - loss: 229361.8125 - mean_squared_error: 229215.5469 - v
al_loss: 92278.3125 - val_mean_squared_error: 92132.4141
Epoch 17/50
338/338 [==============================] - 1s 3ms/step - loss: 227586.3125 - mean_squared_error: 227440.5781 - v
al_loss: 215359.0938 - val_mean_squared_error: 215213.0938
Epoch 18/50
338/338 [==============================] - 1s 3ms/step - loss: 217112.3594 - mean_squared_error: 216965.6406 - v
al_loss: 378406.8125 - val_mean_squared_error: 378260.0938
Epoch 19/50
338/338 [==============================] - 1s 3ms/step - loss: 216314.7656 - mean_squared_error: 216167.3438 - v
al_loss: 169954.2656 - val_mean_squared_error: 169806.5781
Epoch 20/50
338/338 [==============================] - 1s 3ms/step - loss: 215404.7188 - mean_squared_error: 215256.6250 - v
al_loss: 475401.0312 - val_mean_squared_error: 475254.7812
Epoch 21/50
338/338 [==============================] - 1s 3ms/step - loss: 206440.5469 - mean_squared_error: 206294.1719 - v
al_loss: 597649.0000 - val_mean_squared_error: 597505.1250
Epoch 22/50
338/338 [==============================] - 1s 3ms/step - loss: 205770.9219 - mean_squared_error: 205625.4062 - v
al_loss: 237270.8750 - val_mean_squared_error: 237126.4219
Epoch 23/50
338/338 [==============================] - 1s 3ms/step - loss: 201688.4062 - mean_squared_error: 201543.1094 - v
al_loss: 133858.5938 - val_mean_squared_error: 133712.9531
Epoch 24/50
338/338 [==============================] - 1s 3ms/step - loss: 199242.2969 - mean_squared_error: 199096.3750 - v
al_loss: 107264.4062 - val_mean_squared_error: 107118.1719
Epoch 25/50
338/338 [==============================] - 1s 3ms/step - loss: 194240.2812 - mean_squared_error: 194095.6406 - v
al_loss: 96676.3984 - val_mean_squared_error: 96532.4922
Epoch 26/50
338/338 [==============================] - 1s 3ms/step - loss: 191284.3281 - mean_squared_error: 191141.0469 - v
al_loss: 93115.3125 - val_mean_squared_error: 92971.5391
Epoch 27/50
338/338 [==============================] - 1s 3ms/step - loss: 186651.8281 - mean_squared_error: 186508.6562 - v
al_loss: 121231.1172 - val_mean_squared_error: 121087.5391
Epoch 28/50
```

```
338/338 [==============================] - 1s 3ms/step - loss: 182890.4844 - mean_squared_error: 182747.5625 - v
al_loss: 985251.5000 - val_mean_squared_error: 985111.9375
```

Out[ ]: <keras.callbacks.History at 0x7f52fc5307f0>

In [ ]:
```
pred = model5.predict(x_test)
model5.evaluate(x_test, y_test)
```

```
338/338 [==============================] - 1s 2ms/step
338/338 [==============================] - 1s 2ms/step - loss: 92278.3125 - mean_squared_error: 92132.4062
```

Out[ ]: [92278.3125, 92132.40625]

In [ ]:
```
print("Mean Squared error is :", round(mean_squared_error(y_test, pred),2))
print("Mean Absolute error is :", round(mean_absolute_error(y_test, pred),2))
```

```
Mean Squared error is : 92132.42
Mean Absolute error is : 167.6
```